User Guide & Developer Guide for ATROSIM

Viet Hung Nguyen and Le Minh Sang Tran
{ngvhung, minhsang3000}@gmail.com

Introduction: ATROSIM (Another Tool for Routing Protocol Simulation) is an elegant tool that supports simulation of routing protocol in a visual way. This document is a user guide and protocol developer guide for ATROSIM.

1. Main Features

The main features of ATROSIM (also referred to as Simulator), a tool supporting visual routing protocol simulation, are as following:

- Create a network topology that contains switches (nodes) and computers (end-points) by a user-friendly graphical interface.
- The topology can be saved to disk to later usage.
- Specify the connection metric between nodes. The metric could be in uni-direction or bi-direction.
- Able to simulate different protocols.
    - Each protocol is implemented as a separated library and will be plugged into the Simulator at runtime.
    - Protocol developers are able to define data structure for the internal state of each node.
    - Protocol developers are able to use their own message structure to exchange data between nodes.
- Visualize simulation progress
    - Show internal state of each node.
    - Show the input/output messages of a node in a time line (virtual time)
    - Suspend/Resume the routing progress at one or more nodes.
- Visualize protocol reaction
    - Allow user to change topology at runtime to see protocol reaction.

2. Application User-guide

The ATROSIM consists of two major parts: i) Network topology editor; and ii) Protocol Simulation. This part of the document gives an introduction about these functions.

The application needs to run on the .Net 3.5 framework.

How to run the program:

- Execute the file: …*atrosim**ATROSIM**bin**Debug**ATROSIM.exe*
- Or view the source code by open the solution: ...*atrosim**ATROSIM**ATROSIM.sln*, then you can build the solution/project and run it inside the Visual studio.

2.1. Network Topology Editor

The Editor is a kind of Multi-Documents Interface (MDI) editor that allows users to open many documents at the same time; each document is displayed as a tab. At startup, no document is opened; user then can open an existing document (*.atrosim) or create a new one.

2.1.1. How to create a new network document

- Go to menu: *File>New* or click on in the tool bar.
- Create subnet/network:
    - Menu *Windows>Object property*, or hot key F4, to show *Property Window* of this network document as in Figure 1. We strongly suggest you to click on each property in Area 1 to see its description showed in Area 2.

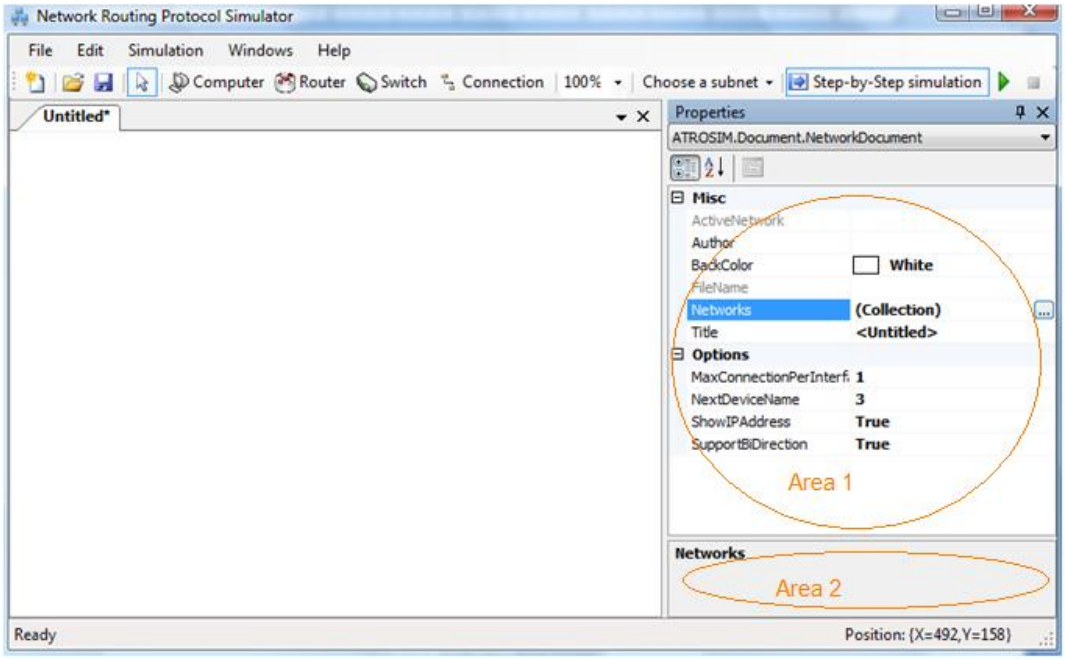


Figure 1: Application Main Screen.

    - In Area 1, click at *Network (Collection)* to open *Network Collection Editor* window as in Figure 2. Then *Add* a subnet and its parameters in the right side, or *Remove* a subnet. Then click *OK.*

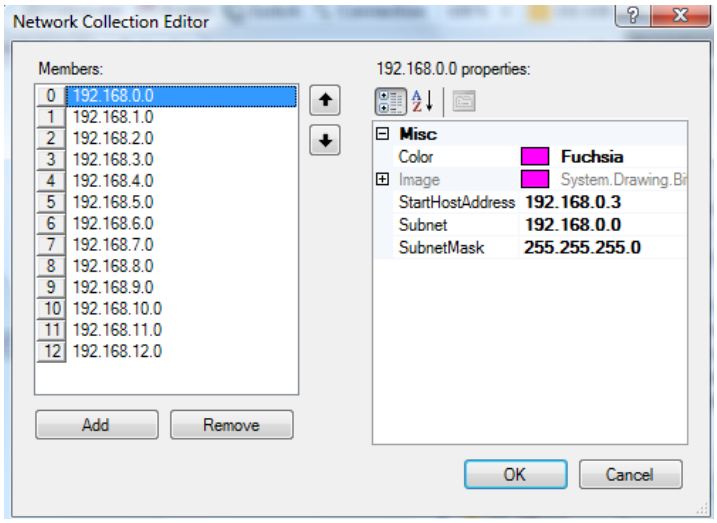
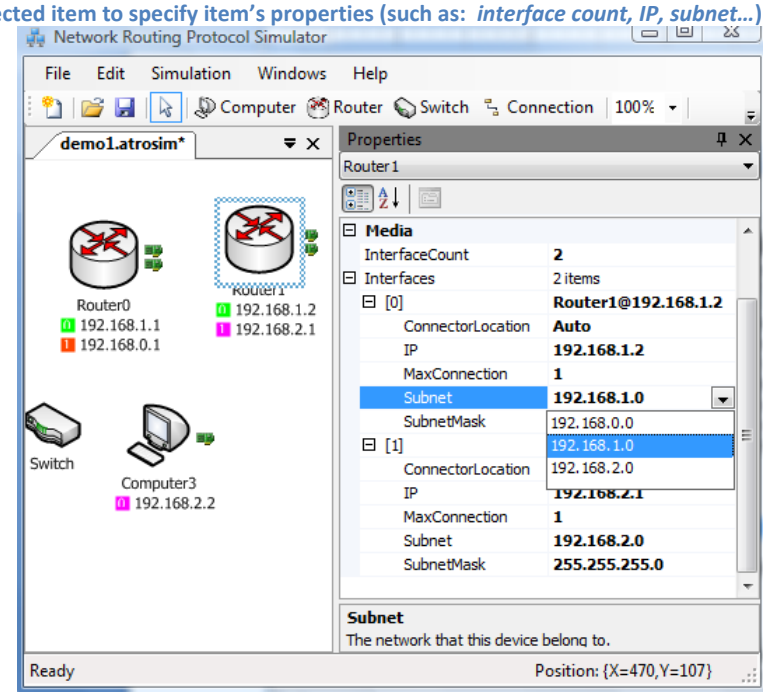


Figure 2: Network Collection Editor window

- Insert new network device (router, computer, switch):
  - Go to menu *Edit>Add Computer, Edit>Add Router, Edit>Add Switch;* or click , , in the tool bar to insert new computer, router, switch.

Select a network device, go to menu *Windows>Object property,* or hot key F4, to show *Property Window* of the selected item to specify item's properties (such as: *interface count, IP, subnet…*) as in



  - Figure 3

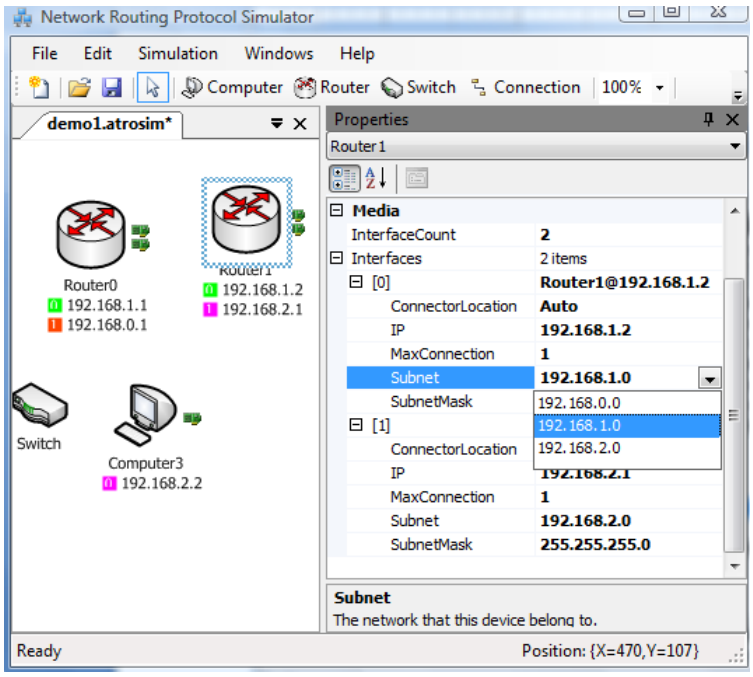


Figure 3: Specify properties of Router 1

- Insert connection:
  - Go to menu *Edit>Add Connection*, or click in tool bar. Then click 2 devices to connect. If target or source device has many interfaces, then window *New Connection* appears as in Figure 4. You can select a source device's interface to connect to a target device's interface.

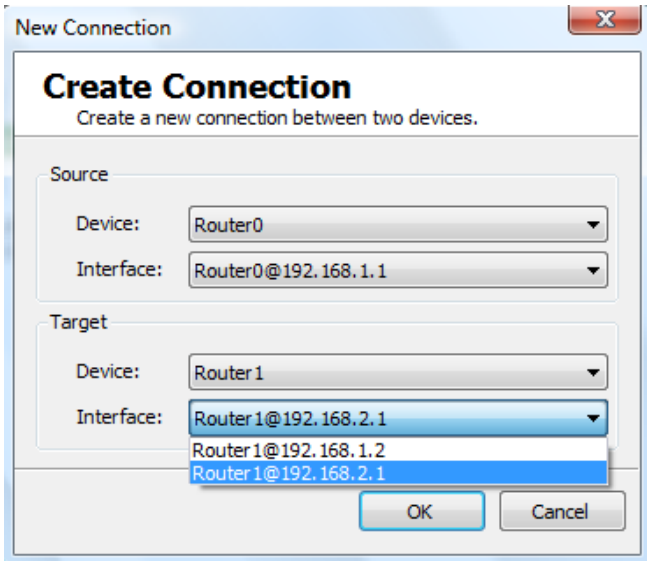


Figure 4: New Connection window

- Edit connection cost: select a connection, go to menu *Windows>Object property,* or hot key F4, to show *Property Window* of the selected connection. Then specify *Cost, ReverseCost* of

connection (note that *ReverseCost* is different from *Cost* only in case *SupportBiDirection = True*). Result will be as in

- .

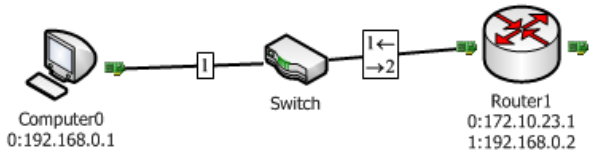


Figure 5: Network topology items and Connection

- Delete an item (router, computer…): go to menu *Edit>Delete* or hot key Delete
- Save file: menu *File>Save*, file extension is *.atrosim.

In general, the following table shows common properties of a topology document and each kind of devices.

TABLE I. LIST OF PROPERTIES APPLIED FOR NETWORK TOPOLOGY OBJECTS.

Property	Cat.	Description
`Networks`		List of subnets in a document. A Subnet has its own color that makes the topology easier to view.
`MaxConnectionPerInterface`		The maximum number of connections for an interface.
`NextDeviceName`		
`ShowIPAddress`		Show/hide IP address below a device.
`SupportBiDirectionCost`		Determine whether this document support uni/bi-direction cost.
`Name`		Name of a device.
`Type`		Type of a device.
`InterfaceCount`		Specify the number of interfaces of a device. For switch, this value always equals one.
`Interfaces`		List of interfaces connected to a device.
`IP`		Determine the IP address of an interface.
`SubnetMask`		Determine the Subnet mask of an interface.
`Subnet`		Determine which subnet a device belongs to. Changing Subnet will change the IP and SubnetMask to appropriate values.
`MaxConnection`		Determine how many connections are able to plug into an interface. This value should not be greater than *MaxConnection-PerInterface*. For switch, this value always equals 2147483647.
`Cost`		Determine traffic cost from Source to Target. Null for disconnected.
`ReverseCost`		Determine traffic cost from Target back to Source. Null for disconnection. When *SupportBiDirectionCost* is false, this value always equals *Cost*.

Legend: - topology document, - computer, - router, - switch, - interface, - connection.

2.1.2. Reuse document's network:

Every time you make a new file, if you feel boring of going to *Network Collection Editor* window, as in Figure 2 , to create a collection of subnets before you can assign which router/computer to which subnet. Then:

- From an existing file (which already has collection of subnets). Go to menu *File>Export document networks…* to save that collection of subnets, file extension: *.network
- When create a new file, you can reuse that collection by going to menu *File>Import document Network…*
- Then, you can click on Choose a subnet on tool bar to choose a particular subnet. Later, every new inserted router/computer will be automatically assign the 1st interface to this particular subnet.

2.2. Protocol Simulation

- Go to menu *Simulation>Options…*, the *Simulation Options* displays. Here you can choose options for the simulation. We strongly suggest choosing *Step-by-step simulation.* You can also click Step-by-Step simulation on tool bar to turn on *step-by-step simulation* mode.
- Go to menu Simulation*>Manage Protocols,* the *Routing Protocol Manager* window is displayed as in Figure 6. In this dialog, you can see the list of available protocols.
    - Click on *Add Library* to add a library (*.dll) containing routing protocol
    - Select a protocol from the list box, click on *Activate* button to make a protocol active
    - Click *Configure* to configure a protocol's parameters if the activated protocol supports this function.

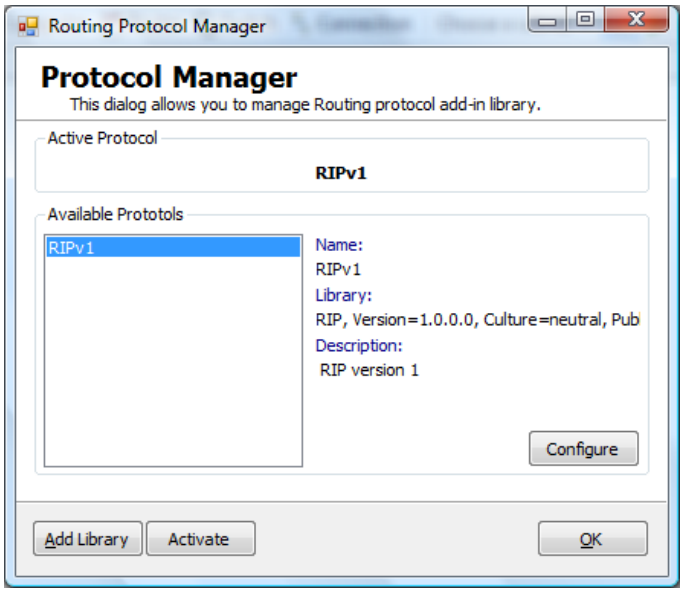


Figure 6: Routing Protocol Manager Dialog

- Go to menu *Simulation>Start* `<Protocol Name>` to begin simulation. If you are in *step-by-step simulation* mode, the simulation will pause after each step [(1)] . Once the simulation pauses, you can see the package flow and state of each node by showing one or more following windows:
    - *Windows>Package Sniffer (F5)*: list of all packages, in time order, sent by the protocol and detail of a package.
    - *Windows>Node State (F6)*: view the newest routing table of selected computer/router. You can also see the internal state of the routing protocol running on this node [(1)].
    - *Windows>Simulation State (F7)*: view the routing table of all nodes in each step of the simulation.
    - *Windows>Simulation Event (F8) (*or *Device Event)*: view the events during the simulation. By default, when a package is sent/receive, a corresponding event is recorded. The protocol can also generate its own event (for instance, when a node runs 'garbage collection' in RIP). You can click on the event to see *its detail* in the *Windows>Property Window (F4)*.
    - *Windows>Console (F9)*: displays all messages emitted during the simulation.

While the simulation is running, you are able to change the network topology to see the protocol reaction, for instance, change connection cost, add device, and so on.

The following are some screen shots illustrating the usage of abovementioned windows. These screen shots are from the simulation of RIP protocol on a document named *test_rip.atrosim.*

[1] This function is available when the simulating protocol gives support.

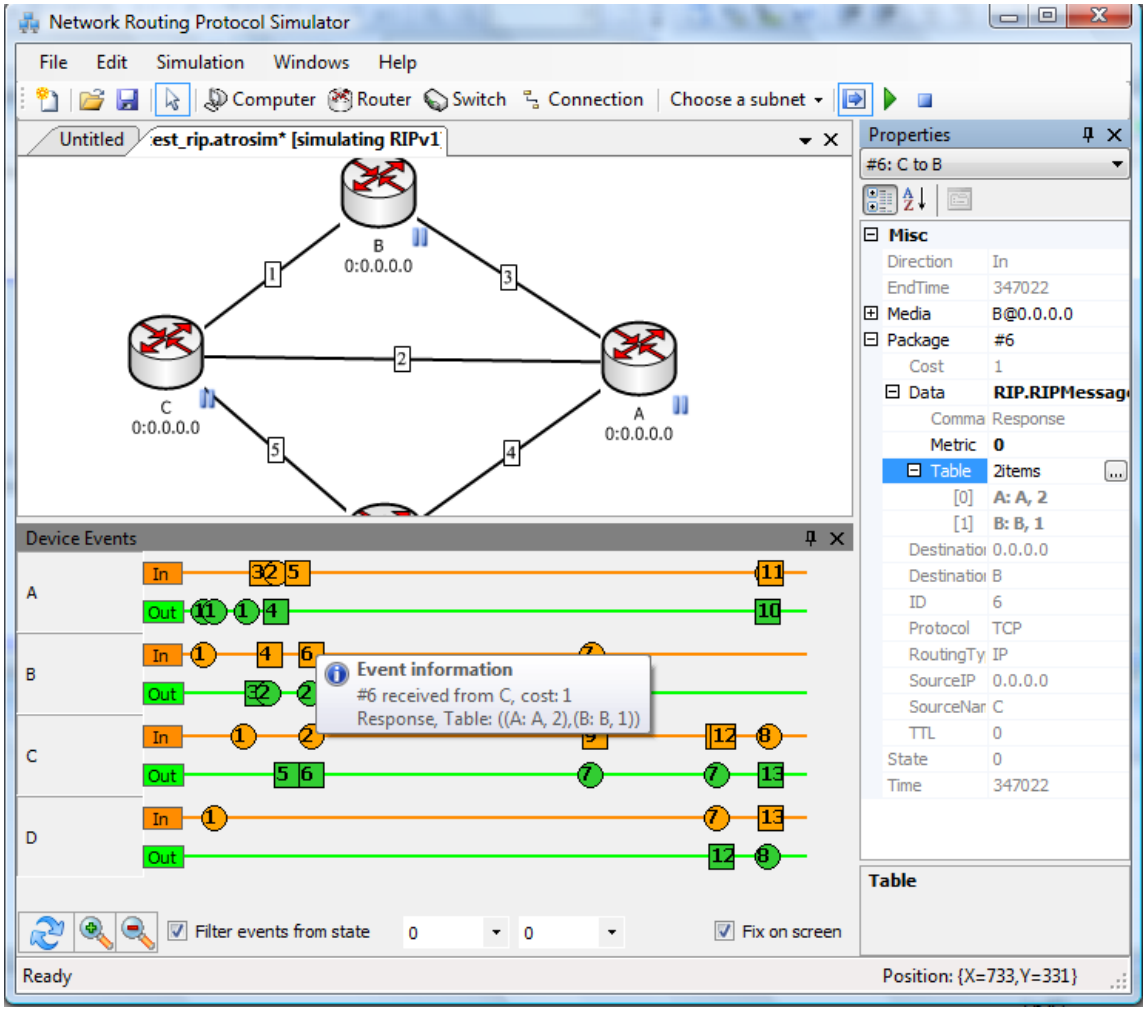


Figure 7: Simulation Event Window and Property Window.

Figure 7 shows the combination of *Windows>Simulation Event* and *Windows>Property Window* to view the traffic flow and detail of an event. In this picture, the cycle(orange) and rectangle(green) represent *Request* and *Response* packages, respectively. These shapes are customized by the RIP protocol.

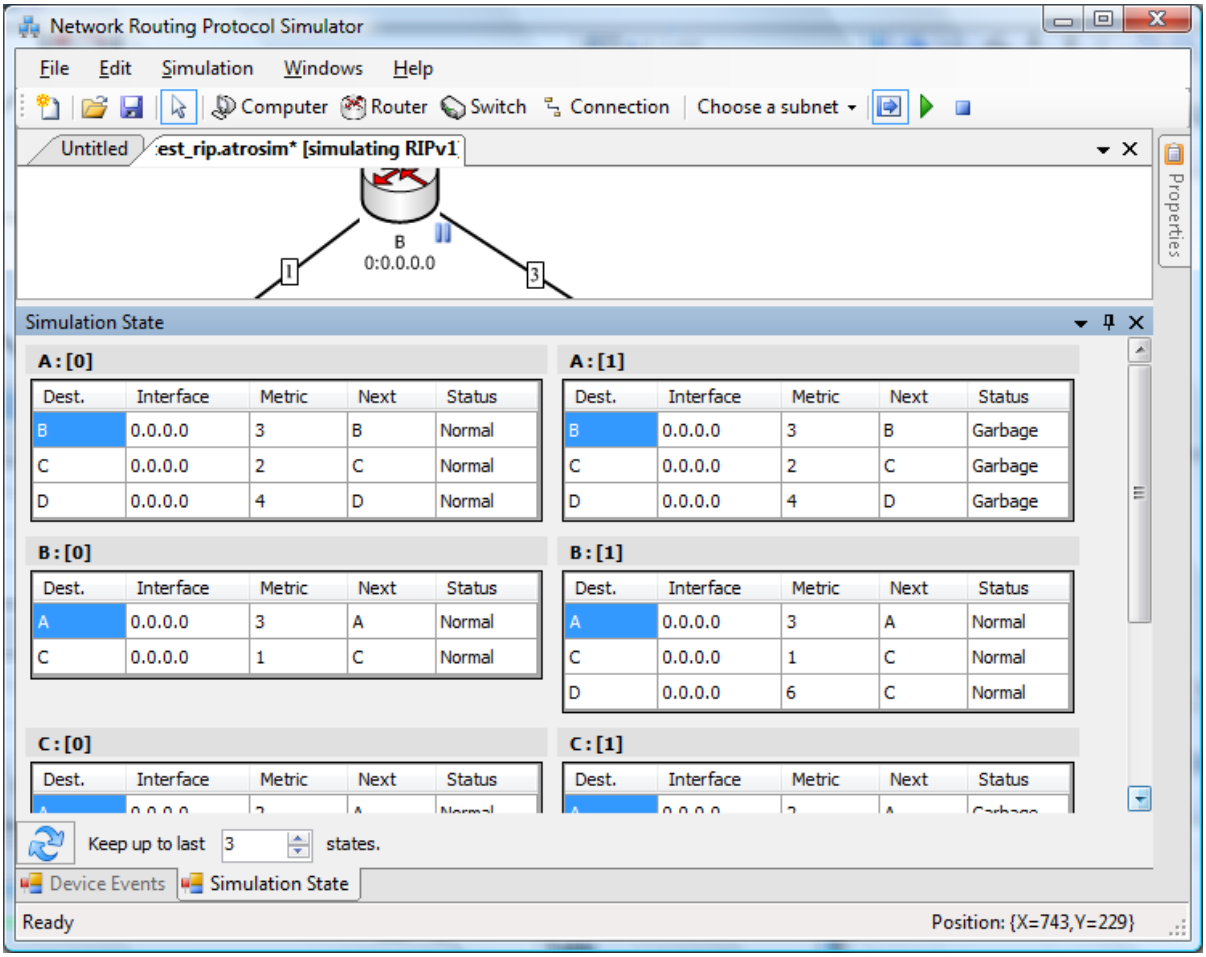


Figure 8: Simulation State Window

Figure 8 shows the usage of *Simulation State* window. The display format is defined by, in this screen shot, the RIP.

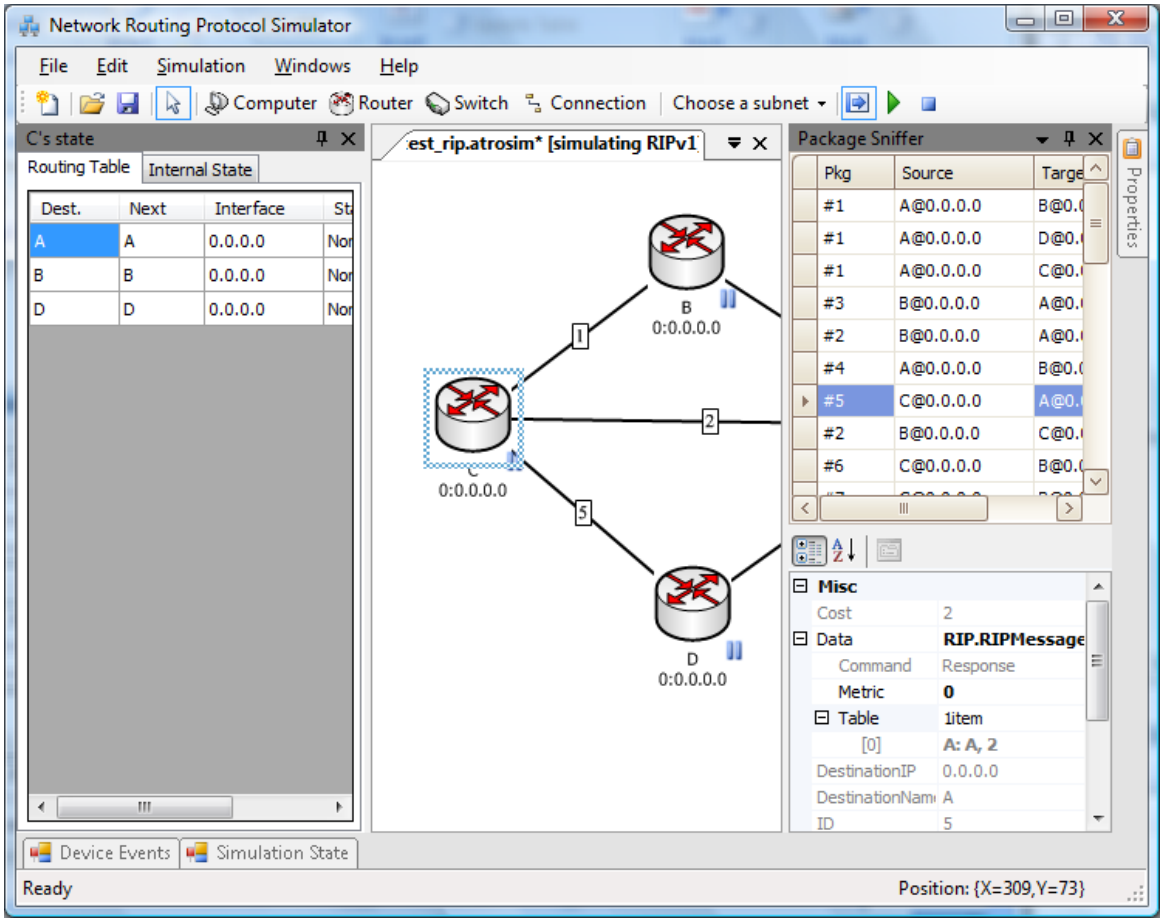


Figure 9: Node State Window and Package Sniffer Window

3. Routing Protocol Plug-in Developer Note

Steps in create a routing protocol library.

1. Create a Class Library Project in Visual Studio 8.
2. Add reference to assembly ATROSIM.exe.
3. Add a class that implements interface ATROSIM.Simulation.IRoutingProtocolController.
4. Add a class that implements interface ATROSIM.Simulation.IRoutingProtocol.
5. Compile the project and enjoy.

If you want to customize the package shape (see Figure 7), then

6. Add a class inherited from ATROSIM.Simulation.PackageEventRecord.
7. Set the PackageEventType property of the ISimulationEngine to your class.

If you want to create your own event, then

8. Add a class inherited from ATROSIM.Simulation.EventRecord.
9. Use the ATROSIM.Simulation.IVisualizer service to record your event.

The detail of IRoutingProtocolController and IRoutingProtocol are described in TABLE II, and TABLE III.

TABLE II: MEMBERS OF IROUTINGPROTOCOLCONTROLLER

	Member Name	Description
	`Name`	Get the name of the protocol.
	`Description`	Get the short description about the protocol.
	`Library`	Get the assembly that the protocol belongs to.
	`Flag`	Get the flag of the protocol, which combines one or more following constants of type `ProtocolFlag`.
	`InternalStateDisplayStyle`	Determine how the internal state is display. It should be a value of type `InternalStateDisplayStyle`.
	`createProtocolInstance`	Create an instance of protocol to run on a node. **Parameters**: - `IAppContext`: current running context. - `RuntimeDevice`: the device that protocol runs on. - `ISimulationEngine`: reference to the simulation engine.
	`createStateViewControl`	Create a control displaying specific state of a protocol. **Parameters**: - `Control`: reference to the container of the created control. - `ProtocolState`: a specific state to view.
	`cloneStateViewControl`	Duplicate a control displaying specific state of a protocol based on an existed one.
	`createInternalStateViewControl`	Create a control displaying the internal state of a protocol. **Parameters**: - `IAppContext`: current running context. - `IRoutingProtocol`: the protocol to be displayed. - `Control`: ref. to container of the control.
	`showOptionDialog`	Display a dialog allowing user to change protocol parameters. **Parameters**: - `IAppContext`: current running context.
	`generatePackageDescription`	Generate a short description for a package like the one display in Figure 7. **Parameters**: - `IAppContext`: current running context. - `StringBuilder`: where you add your description. - `Package`: the package for generating description.

TABLE III: MEMBERS OF IROUTINGPROTOCOL INTERFACE

	Member Name	Description
	`RoutingTable`	Return the current routing table of the protocol.
	`InternalState`	Return the current internal state of the protocol.
	`Step`	Get or set current step of the routing.
	`run`	Called by the simulation engine when it starts the simulation.
	`stop`	Called by the simulation engine when the protocol is being stopped.
	`receiveData`	Called when there is a package sent to this instance. The call is made in a thread separated with that calls the `run`() member. **Parameters**: (filled by the caller) - `MediaInterface`: specify which interface receives data.

	Member Name	Description
		- `Package`: the package sent to this node.
	`findRoute`	Call by the underlying `send`() routine to find out a suitable route. **Parameters:** (filled by the caller) - `HostIdentity` : the identity of the target, in pair of name and IP address. - `[out] uint`: filled by the callee to specify which interface IP used for sending package. - [`out] HostIdentity`: filled by the callee to specify the next destination of sending package. **Return**: true – if a route is found, otherwise, return false.
	`topologyChanged`	Call the simulation engine to inform that the network topology has been changed.
	`lockState`	Called by the visualization engine when it wants to capture the RoutingTable of the protocol.
	`unlockState`	Called by the visualization engine when it finished capturing the RoutingTable of the protocol.
	`pause`	Called by the simulation engine when it tries to suspend the simulation.
	`resume`	Called by the simulation engine when it tries to resume the simulation.

Notice that, all operations in the `IRoutingProtocol` interface are call-back operations. Therefore, you should not make any direct call to them.

The protocol body can take advance of some useful services provided by the simulator, which are listed in TABLE IV. These services are acquired by the given running context. For instance, to get the `IVisualizer` service, you can invoke `Context.getService<IVisualizer>()`.

TABLE IV: SERVICES PROVIDED BY THE APPLICATION CONTEXT

Service Name	Description
ISimulationEngine	Provide access to the simulation engine. Mostly use for internal operations. However, you may find some useful members for developing protocol here.
IVisualizer	Provide means for developer to visualize the running of a protocol.
IConsole	Allows protocol to emit message during its lifetime.

The usage of these services can be found out in an example of implementation of RIP.